

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
ESCUELA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

MANUAL TÉCNICO

“Automatización del proceso de vacunación infantil, parroquia piloto *Cangahua*”.

AUTOR: Reinoso Chicaiza Blanca Lucía

TUTOR: Ing. Irving Reascos

IBARRA-ECUADOR

2015

CONTENIDO

INDICE DE GRÁFICOS

Anexo 1: Manual técnico

En este anexo se explicará de manera más técnica el sistema Informático de vacunación Infantil y el código de programación utilizado.

Objetivo

Puntualizar todas las entidades y código utilizado en cada una de ellas para el funcionamiento del Sistema Informático de Vacunación Infantil *Cangahua* (SIVIC), tomando como lugar piloto al centro de salud de la parroquia de ***Cangahua**.*

Herramientas Utilizadas

- Herramienta de entorno de programación java
  - Eclipse
- Contenedor de Aplicaciones Web
  - Apache Tomcat
- Base de Datos
  - Postgres
- Máquina virtual
  - JDK 1.7
- Creación de Reportes
  - Ireport 5.0

Código Fuente

Raíz del Proyecto, es donde se encuentra el resto del código del Sistema.

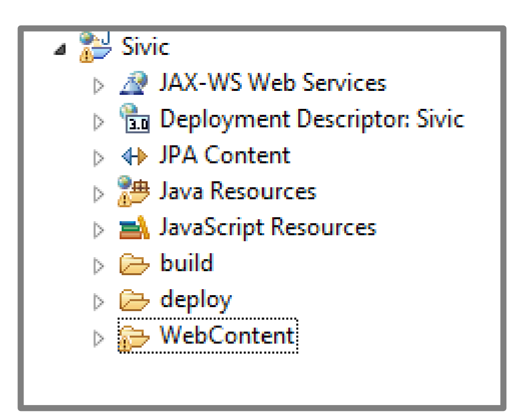


Gráfico 1: Raíz del Sistema SIVIC
Fuente: Propia

Dentro del mismo se compone de las siguientes partes:

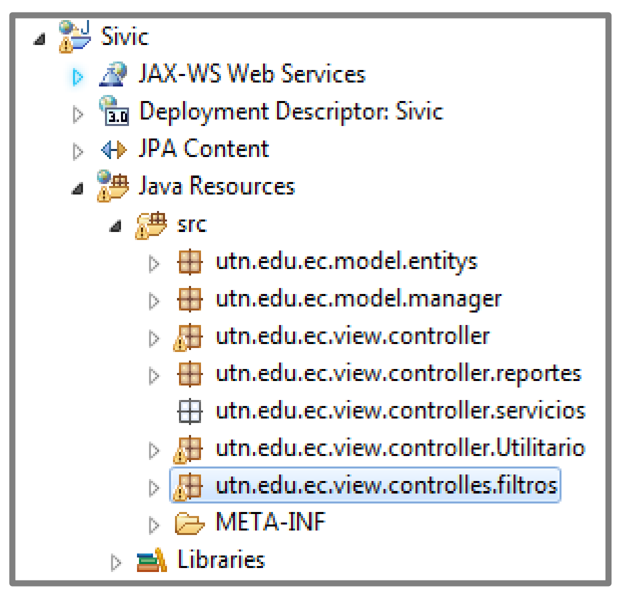


Gráfico 2: Paquetes del sistema
Fuente: Propia

En la carpeta WebContent se encuentra todas las plantillas de vista, archivos de estilo, JavaScript, entre otros que permite la presentación al usuario.

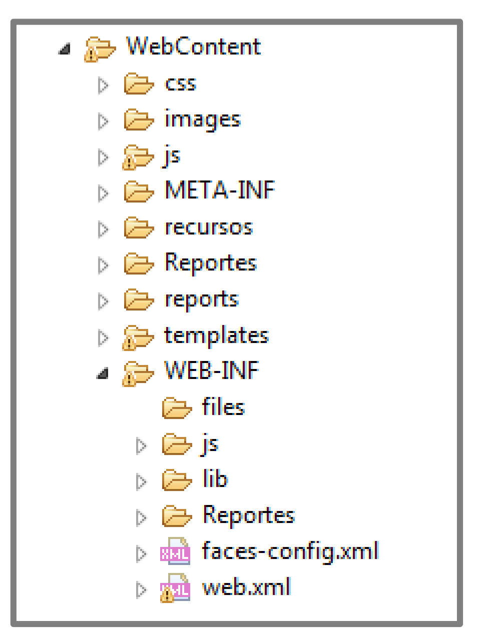


Gráfico 3: Datos del webContent
Fuente: Propia

A continuación se detallará los contenidos que se tiene en cada uno de los paquetes de la carpeta *JavaResources* del proyecto.

JPA

En el primer paquete tenemos las entidades que se lo hace mediante Java Persistence API, esta busca agrupar la manera en que funcionan los servicios que brinda un mapeo objeto relacional, con esto no se pierde la ventaja de orientada a objetos al interactuar con base de datos, además de permitir usar objetos POJO (clases simples java con métodos getter y setter básicos).

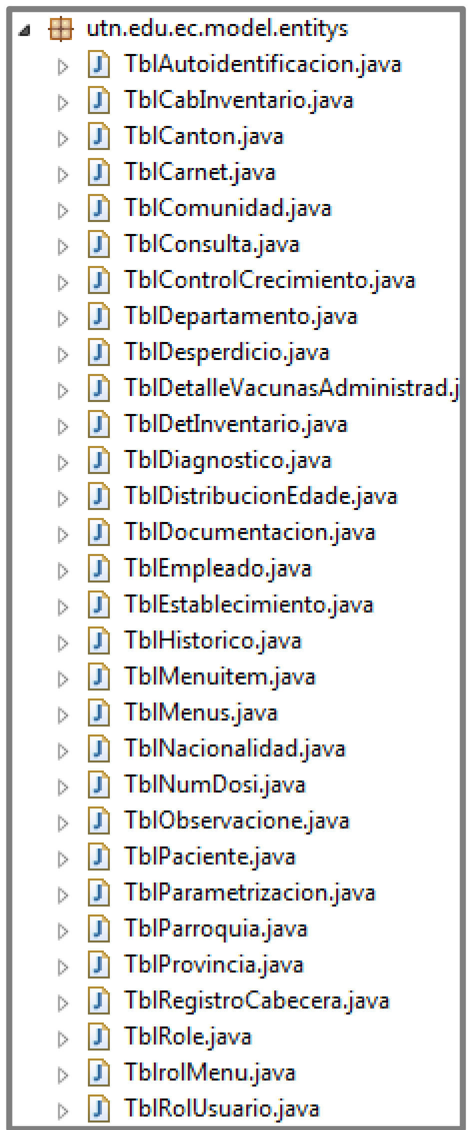


Gráfico 4: Creación de objetos de la BDD con JPA
Fuente: Propia

Todas las entidades que se crea mediante el mapeo tienen la siguiente estructura así que se coloca un ejemplo a continuación:

```
package utn.edu.ec.model.entitys;

import java.io.Serializable;
import javax.persistence.*;
import java.util.List;
```

@Entity.- Los nombres que tienen antepuesto este carácter, declara la clase como una entidad, por ejemplo, una clase POJO (la cual contiene únicamente métodos básicos getters y setters) persistente.

```
@Entity
@Table(name="tbl_departamento")
public class TblDepartamento implements Serializable {
	private static final long serialVersionUID = 1L;
```

Las anotaciones: **@Id** para identificar el identificador de la clase, **@GeneratedValue** para identificar las claves que serán autogeneradas, **@Column** esto es para hacer referencia a la columna de la Base de Datos, **@Temporal** se utiliza para campos de tipo fecha y permite especificar el tipo de temporal.

```
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name="id_departamento", unique=true, nullable=false)
	private Integer idDepartamento;

	@Column(name="nombre_departamento", length=50)
	private String nombreDepartamento;
```

Las anotaciones: **@OneToMany, @ManyToOne,** permiten mapear las relaciones de la base de la datos al crearlos mediante el mapeo.

```
	//bi-directional many-to-one association to TblEmpleado
	@OneToMany(mappedBy="tblDepartamento")
	private List<TblEmpleado> tblEmpleados;
```

El siguiente código es el constructor de la entidad y los métodos getters y setter de las propiedades que se crean de acuerdo al campo de la base de datos.

```
	public TblDepartamento() {
	}

	public Integer getIdDepartamento() {
		return this.idDepartamento;
	}

	public void setIdDepartamento(Integer idDepartamento) {
		this.idDepartamento = idDepartamento;
	}

	public String getNombreDepartamento() {
		return this.nombreDepartamento;
```

```java
	}

	public void setNombreDepartamento(String nombreDepartamento) {
		this.nombreDepartamento = nombreDepartamento;
	}

	public List<TblEmpleado> getTblEmpleados() {
		return this.tblEmpleados;
	}

	public void setTblEmpleados(List<TblEmpleado> tblEmpleados) {
		this.tblEmpleados = tblEmpleados;
	}

	public TblEmpleado addTblEmpleado(TblEmpleado tblEmpleado) {
		getTblEmpleados().add(tblEmpleado);
		tblEmpleado.setTblDepartamento(this);

		return tblEmpleado;
	}

	public TblEmpleado removeTblEmpleado(TblEmpleado tblEmpleado) {
		getTblEmpleados().remove(tblEmpleado);
		tblEmpleado.setTblDepartamento(null);

		return tblEmpleado;
	}

}
```

Utn.edu.ec.model.manager: En el siguiente paquete tiene todos los manejadores básicos los cuales permite interactuar con la base de datos. Los métodos más importantes que aquí se tienen son:

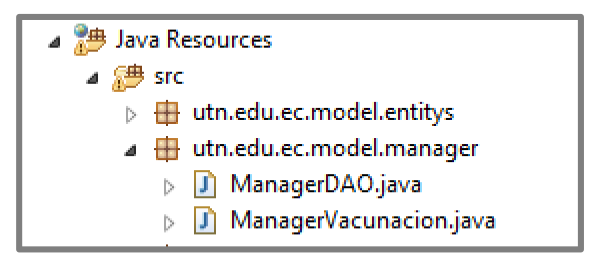


Gráfico 5: Contenido del paquete model.manager
Fuente: Propia

- Insertar
- Eliminar
- Actualizar
- Buscar Por ID
- Buscar por alguna campo en específico

- ➢ Listas Filas de una tabla de la base de datos
- ➢ Listar datos de acuerdo a un ID

A continuación código de los manejadores básicos:

```
/**
 * finder Generico que devuelve todos las entidades de una tabla.
 *
 * @param clase
 *           La clase que se desea consultar.
 * @param orderBy
 *           Propiedad de la entidad por la que se desea ordenar la
 *           consulta.
 * @return Listado resultante.
 */
@SuppressWarnings("rawtypes")
public List findAll(Class clase, String orderBy) {
      mostrarLog("findAll: " + clase.getSimpleName() + " orderBy " +
      orderBy);
      Query q;
      List listado = null;
      try {
            if (!em.getTransaction().isActive()) {
                  em.getTransaction().begin();
                  mostrarLog("transaccion begin");
            }
            if (orderBy == null || orderBy.length() == 0)
                  q = em.createQuery("SELECT o FROM " +
                  clase.getSimpleName()+ " o");
            else
                  q = em.createQuery("SELECT o FROM " +
                  clase.getSimpleName()+ " o ORDER BY " + orderBy);
            listado = q.getResultList();
            if (em.getTransaction().isActive()) {
                  em.getTransaction().commit();
            }
      } catch (Exception e) {
            FacesContext.getCurrentInstance().addMessage(null,
                        new FacesMessage("Problemas al listar Objetos
                        ordenado"));
            }

      return listado;
}
/ * @param clase
 *           La clase que se desea consultar.
 * @param orderBy
 *           Propiedad de la entidad por la que se desea ordenar la
 *           consulta.
 * @return Listado resultante.
 */
@SuppressWarnings("rawtypes")
public List findAllT(Class clase, String orderBy) {
```

```java
        mostrarLog("findAllT: " + clase.getSimpleName() + " orderBy " +
        orderBy);
        Query q;
        List listado;
        if (!em.getTransaction().isActive()) {
                em.getTransaction().begin();
                mostrarLog("transaccion begin");
        }
        if (orderBy == null || orderBy.length() == 0)
                q = em.createQuery("SELECT o FROM " + clase.getSimpleName() + "
                o");
        else
                q = em.createQuery("SELECT o FROM " + clase.getSimpleName()
                                + " o ORDER BY " + orderBy);
        listado = q.getResultList();
        /*
        * if(!em.getTransaction().isActive()) em.getTransaction().commit();
         */
        return listado;
}

/* @param clase
 *              La clase que se desea consultar.
 * @return devuelve un Listado como resultado.
 */
@SuppressWarnings("rawtypes")
public List findAll(Class clase) {
        mostrarLog("findAll: " + clase.getSimpleName());
        Query q;
        List listado = null;
        try {
                if (!em.getTransaction().isActive())
                        em.getTransaction().begin();
                q = em.createQuery("SELECT o FROM " + clase.getSimpleName() + "
                o");
                listado = q.getResultList();
                em.getTransaction().commit();
        } catch (Exception e) {
                FacesContext.getCurrentInstance().addMessage(null,
                                new FacesMessage("Problemas al listar Objetos"));
        }

        return listado;
}
/* @param clase
 *  La entidad sobre la que se desea consultar.
 * @param pClausulaWhere
  * Clausula where de tipo JPQL (sin la palabra reservada WHERE).
 *  @param claa es el nombre de la clase a la cual pertenece una determinada
 clave foranea
 * @param p el valor que ingresa el usuario por el cual desea buscar datos
 * @param campo es como se llama el campo por el cual se va ha buscar
 * @return valor devuelto es Listado.
 */
@SuppressWarnings("rawtypes")
```

```
public List findWhere(Class clase, String claa, Integer p, String campo){
       mostrarLog("findAll(where): " + clase.getSimpleName() + " Valor de
       p:"+ p + " Valor de campo :" + campo + " Valor de clase.>>>" +  claa);
       Query q;
       List listado;
       if (!em.getTransaction().isActive())
              em.getTransaction().begin();
              q = em.createQuery("SELECT o FROM " + clase.getSimpleName()
                     + " o WHERE o." + claa + "." + campo + " = :" + campo);
       q.setParameter(campo, p);
       listado = q.getResultList();
       em.getTransaction().commit();
       return listado;
}
/*@param clase la entidad sobre la que se desea consultar
 * @param p el valor que ingresa el usuario por el cual desea buscar datos
 * @param campo es como se llama el campo por el cual se va a buscar
 * @return valor devuelto es Listado.
 * */
@SuppressWarnings("rawtypes")
public List findWhereTa(Class clase, Integer p, String campo) {
       mostrarLog("findAll(where): " + clase.getSimpleName() + " Valor de p:"
       + p + " Valor de campo :" + campo + " Valor de clase.>>>");
       Query q;
       List listado;
       if (!em.getTransaction().isActive())
              em.getTransaction().begin();
       q = em.createQuery("SELECT o FROM " + clase.getSimpleName()
                     + " o WHERE o." + campo + " = :" + campo);
       q.setParameter(campo, p);
       listado = q.getResultList();
       em.getTransaction().commit();
       return listado;
}

/* método que busca de acuerdo a un id foránea de una tabla y solo retorna
un objeto
@param clase
 *  La entidad sobre la que se desea consultar.
 *  @param claa es el nombre de la clase a la cual pertenece una determinada
 clave foranea
 * @param p el valor que ingresa el usuario por el cual desea buscar datos
 * @param campo es como se llama el campo por el cual se va ha buscar
 * @return valor devuelto es un solo objeto.
 */

@SuppressWarnings("rawtypes")
public Object findWhereOb(Class clase, String claa, Integer p, String campo)
{
       mostrarLog("findAll(where): " + clase.getSimpleName() + " Valor de
       p:"+ p + " Valor de campo :" + campo + " Valor de clase.>>>"
                     + claa);
       Query q;
       Object listado;
```

```
        if (!em.getTransaction().isActive())
            em.getTransaction().begin();
        q = em.createQuery("SELECT o FROM " + clase.getSimpleName()
                + " o WHERE o." + claa + "." + campo + " = :" + campo);
        q.setParameter(campo, p);
        // SELECT c FROM Country c WHERE c.population > :p
        listado = q.getSingleResult();
        em.getTransaction().commit();
        return listado;
}
/* @param clase
 *            La clase sobre la que se desea consultar.
 * @param pID
 *            Identificador que permitirá la búsqueda.
 * @return El objeto solicitado (si existiera).
 * @throws Exception
 */
@SuppressWarnings({ "rawtypes", "unchecked" })
public Object findById(Class clase, Object pID) throws Exception {
        Object o;
        try {
            if (!em.getTransaction().isActive())
                em.getTransaction().begin();
            if (pID == null)
                throw new Exception(
                        "Debe especificar el cï¿½digo para buscar el
                        dato.");

            o = em.find(clase, pID);
            em.getTransaction().commit();
        } catch (Exception e) {
            em.getTransaction().rollback();
            throw new Exception("No se encontró el dato especificado: ");
        }

        return o;
}
/**
 * Almacena un objeto en la persistencia.
 *
 * @param pObjeto
 *            El objeto a insertar.
 * @throws Exception
 */
public void insertar(Object pObjeto) throws Exception {
        System.out.println("Manager DAO Lucia de ingreso>>>");
        if (!em.getTransaction().isActive())
            em.getTransaction().begin();
        try {
            em.persist(pObjeto);
            mostrarLog("Objeto insertado: "
                    + pObjeto.getClass().getSimpleName());
            em.getTransaction().commit();
        } catch (Exception e) {
```

```java
            FacesContext.getCurrentInstance().addMessage(null,
                    new FacesMessage("Error al insertar Objeto"));
        }

    }
    /**
     * Elimina un objeto de la persistencia.
     *
     * @param clase
     *            La clase correspondiente al objeto que se desea eliminar.
     * @param pID
     *            El identificador del objeto que se desea eliminar.
     * @throws Exception
     */
    @SuppressWarnings("rawtypes")
    public void eliminar(Class clase, Object pID) throws Exception {
        if (pID == null) {
            mostrarLog("Debe especificar un identificador para eliminar el
            dato solicitado: "+ clase.getSimpleName());
            throw new Exception(
        "Debe especificar un identificador para eliminar el dato
        solicitado.");
        }
        Object o = findById(clase, pID);
        if (!em.getTransaction().isActive())
            em.getTransaction().begin();
        try {
            em.remove(o);
            mostrarLog("Dato eliminado: " + clase.getSimpleName() + " "
                    + pID.toString());
            em.getTransaction().commit();
        } catch (Exception e) {
            FacesContext
                    .getCurrentInstance()
                    .addMessage(
                            null,
                            new FacesMessage(
            "Problema al aliminar el objeto con el id : "
        + pID+ "  Puede estar Refereciada en otra Tabla"));
        }

    }


    /**
     * Actualiza la información de un objeto en la persistencia.
     *
     * @param pObjeto
     *Objeto que contiene la información que se debe actualizar.
     * @throws Exception
     */
    public void actualizar(Object pObjeto) throws Exception {
        if (pObjeto == null)
            throw new Exception("No se puede actualizar un dato null");
```

```java
        if (!em.getTransaction().isActive())
            em.getTransaction().begin();
        try {
            em.merge(pObjeto);
            mostrarLog("Dato actualizado: "
                        + pObjeto.getClass().getSimpleName());
            em.getTransaction().commit();
        } catch (Exception e) {
            FacesContext.getCurrentInstance().addMessage(null,
                        new FacesMessage("Problema al Actualizar el
            dato"));
        }

    }


    /**
     *permite obtener el máximo id de una determinada tabla.
     *
    * @param clase es el nombre de la tabla que deseamos obtener el máximo id
     * @param campo es el nombre de la columna que desea obtener el máximo id
     * @throws Exception
     */
    @SuppressWarnings("rawtypes")
    public Object ObtenerIdMaxLista(Class clase, String campo) throws Exception
    {
        System.out.println("Ingreso al metodo del Manager DAO");
        Object valor = "";
        try {
            Query q;
            if (!em.getTransaction().isActive())
                em.getTransaction().begin();
            q = em.createQuery("SELECT MAX(o." + campo + ") FROM "
                        + clase.getSimpleName() + " o
            valor = q.getSingleResult();
            em.getTransaction().commit();
        } catch (Exception e) {
            FacesContext.getCurrentInstance().addMessage(null,
                        new FacesMessage("Problema al obtener el dato
                solicitado"));
        }

        return valor;
    }
// MÉTODO PARA BUSCAR LAS HISTORIAS DE USUARIO QUE AÚN NO TIENE CARNE
    @SuppressWarnings("rawtypes")
    public List findWhereObjeto(Class clase, String campo, Object p) {
        mostrarLog("findAllObjeto: " + clase.getSimpleName() + " Valor de p:"
        + p + " Valor de campo :" + campo);
        Query q;
        List listado = null;
        try {
            if (!em.getTransaction().isActive())
                em.getTransaction().begin();
```

```java
                q = em.createQuery("SELECT o FROM " + clase.getSimpleName()
                        + " o WHERE o." + campo + " = :" + campo);
                q.setParameter(campo, p);
                // SELECT c FROM Country c WHERE c.population > :p
                listado = q.getResultList();
                em.getTransaction().commit();
            } catch (Exception e) {
                FacesContext.getCurrentInstance().addMessage(null,
                        new FacesMessage("No existe Objeto con el id
                    buscado"));
            }
            return listado;
        }
```

A continuación se mostrará la siguiente entidad que se tiene la cual es la intermediaria entre el Manejador de objetos y el controlador, en este caso se le llama ManagedVacunacion, se utilizan todos los manejadores básicos que se crearon anteriormente para todas las entidades que se tiene en JPA, en este caso se colocará de una entidad en particular pero el resto todos manejan lo mismo.

```java
package utn.edu.ec.model.manager;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import javax.faces.model.SelectItem;
import utn.edu.ec.model.entitys.TblAutoidentificacion;
import utn.edu.ec.model.entitys.TblCanton;
import utn.edu.ec.model.entitys.TblCarnet;
import utn.edu.ec.model.entitys.TblComunidad;
import utn.edu.ec.model.entitys.TblConsulta;
import utn.edu.ec.model.entitys.TblControlCrecimiento;
import utn.edu.ec.model.entitys.TblDepartamento;
import utn.edu.ec.model.entitys.TblDesperdicio;
import utn.edu.ec.model.entitys.TblDetalleVacunasAdministrad;
import utn.edu.ec.model.entitys.TblDistribucionEdade;
import utn.edu.ec.model.entitys.TblEmpleado;
import utn.edu.ec.model.entitys.TblEstablecimiento;
import utn.edu.ec.model.entitys.TblHistorico;
import utn.edu.ec.model.entitys.TblMenuitem;
import utn.edu.ec.model.entitys.TblMenus;
import utn.edu.ec.model.entitys.TblNacionalidad;
import utn.edu.ec.model.entitys.TblNumDosi;
import utn.edu.ec.model.entitys.TblPaciente;
import utn.edu.ec.model.entitys.TblParroquia;
import utn.edu.ec.model.entitys.TblProvincia;
import utn.edu.ec.model.entitys.TblRegistroCabecera;
import utn.edu.ec.model.entitys.TblRolUsuario;
import utn.edu.ec.model.entitys.TblRole;
import utn.edu.ec.model.entitys.TblStockVacuna;
```

```java
import utn.edu.ec.model.entitys.TblSubmenus;
import utn.edu.ec.model.entitys.TblUnionDistribucionVacunaN;
import utn.edu.ec.model.entitys.TblUsuario;
import utn.edu.ec.model.entitys.TblVacunaDosi;
import utn.edu.ec.model.entitys.TblVacunadosisConsulta;
import utn.edu.ec.model.entitys.TblVacunasCarnet;
import utn.edu.ec.model.entitys.TblrolMenu;

public class ManagerVacunacion {
	private ManagerDAO mDAO = new ManagerDAO();
	// Constructor de la clase estudiada
	public ManagerVacunacion() {

	}

	// MÉTODOS DE DEPARTAMENTO
	// Método listar los departamentos médicos del centro de salud
	@SuppressWarnings("unchecked")
	public List<TblDepartamento> findAllDepartamento() {
		return mDAO.findAll(TblDepartamento.class, "o.idDepartamento");
	}
	// Métodos ingresar departamento médico
	public void IngresarDepartamento(TblDepartamento c) throws Exception {
		mDAO.insertar(c);
	}
	// Métodos eliminar departamento médico
	public void EliminarDepartamento(int det) throws Exception {
		mDAO.eliminar(TblDepartamento.class, det);
	}
	// Métodos Actualizar departamento médico
	public void ActualizarDepartamento(TblDepartamento c) throws Exception {
		mDAO.actualizar(c);
	}
	// Métodos buscar departamento médico dado un id de parámetro
	public TblDepartamento findDepartamentoById(int cod) throws Exception {
		return (TblDepartamento) mDAO.findById(TblDepartamento.class, cod);
	}
}
```

utn.edu.ec.view.controller.- Es el encargado del control de toda la presentación, aquí se consumen los métodos realizados anteriormente, es por ello que se dice que es el intermediario entre el Manejador de Objetos y el controlador, con esto podemos hacer al código mucho más reutilizable ya que de cualquier entidad controladora podemos llamar a los mismos métodos de crear departamento médico por ejemplo.

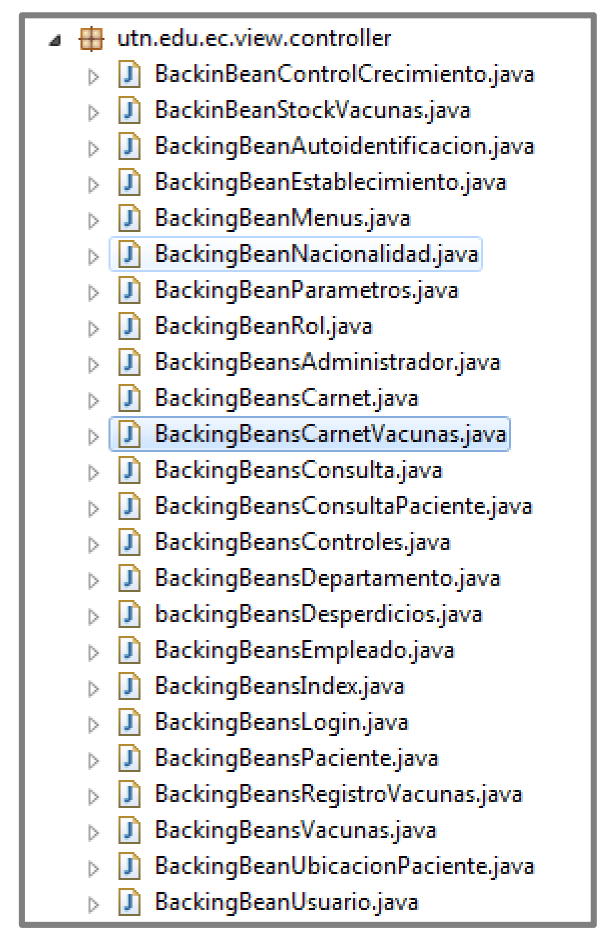


Gráfico 6: Beans del sistema
Fuente: Propia

Estas clases básicas conocidas como POJO, permiten manejar y controlar el sistema mediante el agregado `@ManagedBean, @SessionScoped` en dichas clases y mediante el manejo de lenguaje de expresiones, debido a que todas las clases maneja la misma estructura se colocará un ejemplo:

```
package utn.edu.ec.view.controller;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.servlet.ServletContext;
```

```java
import javax.servlet.http.HttpServletResponse;

import org.primefaces.component.datatable.DataTable;
import org.primefaces.event.RowEditEvent;

import view.controller.Utilitario.Operaciones;

import com.itextpdf.text.Document;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfWriter;

import utn.edu.ec.model.entitys.TblDepartamento;
import utn.edu.ec.model.manager.ManagerVacunacion;
```

@ManagedBean.- Ayuda a identificar y acceder desde el diseño, además permiten registrar las clases que van a ser Managed Beans. La anotación es @javax.faces.bean.ManagedBean, con esto ya no tenemos que especificar esta información en el archivo XML faces-config.xml, que tiene el proyecto creado.

@SessionScoped.- Nos permite manejar todo el proceso únicamente hasta cuando se termine la sesión actual.

```java
@ManagedBean
@SessionScoped
public class BackingBeansDepartamento {
	// Variables del manejador intermediario entre el manejador de Objetos y el controlador
	private ManagerVacunacion mVacunacion = new ManagerVacunacion();
	private Integer codigo_dep;
	private String nombre_dep;
	private TblDepartamento dep;
	private DataTable tablaDepartamento;
	// Variables para filtros general
	private List<TblDepartamento> lista_dep;

	// VARIABLES DE REPORTE
	private List<TblDepartamento> listaObjetos;

	@ManagedProperty(value = "#{backingBeansLogin}")
	private BackingBeansLogin miLogin;
	private Operaciones ope;

	public BackingBeansDepartamento() {
		dep = new TblDepartamento();
		ope = new Operaciones();
	}
	// métodos getters y setters
	public TblDepartamento getDep() {
		return dep;
	}
```

```java
	public void setDep(TblDepartamento dep) {
		this.dep = dep;
	}
	public DataTable getTablaDepartamento() {
		return tablaDepartamento;
	}

	public void setTablaDepartamento(DataTable tablaDepartamento) {
		this.tablaDepartamento = tablaDepartamento;
	}

	public Integer getCodigo_dep() {
		return codigo_dep;
	}

	public void setCodigo_dep(Integer codigo_dep) {
		this.codigo_dep = codigo_dep;
	}

	public String getNombre_dep() {
		return nombre_dep;
	}

	public void setNombre_dep(String nombre_dep) {
		this.nombre_dep = nombre_dep;
	}

	public List<TblDepartamento> getListaDepartamento() {
		return mVacunacion.findAllDepartamento
	}

	public BackingBeansLogin getMiLogin() {
		return miLogin;
	}
	public void setMiLogin(BackingBeansLogin miLogin) {
		this.miLogin = miLogin;
	}

	//Filtros general
	public List<TblDepartamento> getLista_dep() {
		return lista_dep;
	}

	public void setLista_dep(List<TblDepartamento> lista_dep) {
		this.lista_dep = lista_dep;
	}


// MÉTODOS PARA EL FUNCIONAMIENTO DEL FORMULARIO DEPARTAMENTO MÉDICO
	//este método permite limpiar el formulario para crear nuevo departamento
	médico
	public void AccionNuevo() {
		nombre_dep = "";
	}
```

```java
	//método que permite Insertar un nuevo Departamento Médico con todos sus
	atributos
	public String AccionInsertarDepartamento() {
		FacesMessage msg = null;
		String men = "";
		//en la siguiente linea capturamos la session del login para poder
		//ver el usuario que está realizando dicha operación
		miLogin = (BackingBeansLogin) ope.AccionCapturar().getAttribute(
				"backingBeansLogin");
		dep = new TblDepartamento();
		dep.setNombreDepartamento(nombre_dep);
		try {
			mVacunacion.IngresarDepartamento(dep); // ingresamos el
			departamento llamando
			//al método que esta creado en el manejador intermediario
			// En la siguiente línea enviamos los datos a la tabla
			histórico para mantener
			//información de quien está realizando las operaciones

	mVacunacion.IngresarHistorico(miLogin.getUsua().getIdUsuario(), "",
					nombre_dep, "INSERT", ope.obtenerIp(),
				"TblDepartamento");
			men = "Guardado Correctamente";
			msg = new FacesMessage(FacesMessage.SEVERITY_INFO, "Mensaje :
			",men);
		} catch (Exception e) {
			men = "No se puedo Guardar Datos..!!";
			msg = new FacesMessage(FacesMessage.SEVERITY_ERROR, "Error : ",
			men);
		}
		FacesContext.getCurrentInstance().addMessage(null, msg);
		return "";
	}

//método que permite eliminar un departamento esto en caso
	//de haber creado un departamento por error
	public String AccionEliminarDepartamento(TblDepartamento p) {
		FacesMessage msg = null;
		String men = "";
		try {
			// En la siguiente línea enviamos los datos a la tabla
			histórico para mantener
			//información de quien está realizando las operaciones antes de
			que este sea eliminado
		mVacunacion.IngresarHistorico(miLogin.getUsua().getIdUsuario(),
		"Id Departamento ::" + p.getIdDepartamento() + "  Detalle::" +
		p.getNombreDepartamento(), "", "DELETE", ope.obtenerIp(),
		"TblDepartamento");
		//procedemos a eliminar el departamento médico que ha sido
		seleccionado
			mVacunacion.EliminarDepartamento(p.getIdDepartamento());
			men = "Borrado Correctamente";
			msg = new FacesMessage(FacesMessage.SEVERITY_INFO, "Mensaje :
			",men);
		} catch (Exception e) {
```

```
                men = "Erro al tratar de Eiminar..!!";
                msg = new FacesMessage(FacesMessage.SEVERITY_ERROR, "Error : ",
                men);
        }
        FacesContext.getCurrentInstance().addMessage(null, msg);
        return "";
}

//controlador que permite mostrar la lista de todos los
//departamentos médicos que existe en la base de datos
public List<TblDepartamento> getListaDep() {
        return mVacunacion.findAllDepartamento();
}

// Método para capturar el objeto cuando se edita con el elemento de
// primefaces
public void rowEditListener(RowEditEvent event) {
        TblDepartamento depar = (TblDepartamento) event.getObject();
        try {

mVacunacion.IngresarHistorico(miLogin.getUsua().getIdUsuario(), "",
        depar.getNombreDepartamento(), "UPDATE", ope.obtenerIp(),
        "TblDepartamento");
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        ope.MSGGuardarCambios(depar.getNombreDepartamento());
}
```

utn.edu.ec.view.controller.reportes .- En este paquete se tienen todos los manejadores que tienen la producción de reportes del sistema.

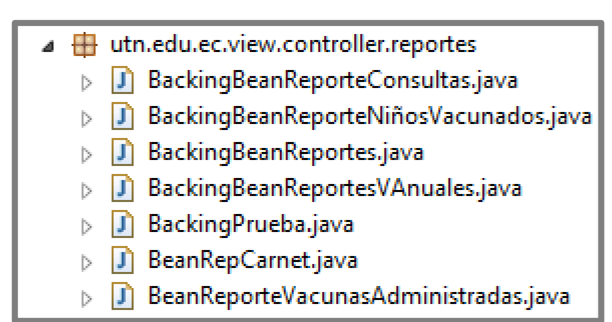


Gráfico 7: Contenido del paquete de reportes
Fuente: Propia

En el Gráfico 51, tenemos los beans que permiten en manejo de reportes como en todos se maneja de la misma manera detallare el reporte de la cantidad de niñas y niños vacunados en los años que seleccione el usuario:

```
package utn.edu.ec.view.controller.reportes;
```

```java
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.faces.model.SelectItem;

import org.primefaces.event.ItemSelectEvent;
import org.primefaces.model.chart.CartesianChartModel;
import org.primefaces.model.chart.ChartSeries;
import org.primefaces.model.chart.PieChartModel;

import utn.edu.ec.model.entitys.TblCarnet;
import utn.edu.ec.model.entitys.TblPaciente;
import utn.edu.ec.model.entitys.TblVacunadosisConsulta;
import utn.edu.ec.model.entitys.TblVacunasCarnet;
import utn.edu.ec.model.manager.ManagerVacunacion;
@ManagedBean
@SessionScoped
public class BackingBeanReporteNiñosVacunados {
	private ManagerVacunacion mVacunacion = new ManagerVacunacion();
	// Variables utilizadas para la generación de reportes
	private String nombreVacuna;
	private List<TblPaciente> lista_pac;
	private Integer cod_vc;
	private String detalle_edad;
	private Date fecha_vacunas;
	private PieChartModel estadisticaGraf;
	private CartesianChartModel repBarra;

	// Método constructor
	public BackingBeanReporteNiñosVacunados() {
		// TODO Auto-generated constructor stub
	}

	// Métodos accesores (getters y setters)
	public String getNombreVacuna() {
		return nombreVacuna;
	}

	public void setNombreVacuna(String nombreVacuna) {
		this.nombreVacuna = nombreVacuna;
	}

	public List<TblPaciente> getLista_pac() {
		return lista_pac;
	}
	public void setLista_pac(List<TblPaciente> lista_pac) {
		this.lista_pac = lista_pac;
	}
```

```java
public Integer getCod_vc() {
	return cod_vc;
}

public void setCod_vc(Integer cod_vc) {
	this.cod_vc = cod_vc;
}

public String getDetalle_edad() {
	return detalle_edad;
}

public void setDetalle_edad(String detalle_edad) {
	this.detalle_edad = detalle_edad;
}

public Date getFecha_vacunas() {
	return fecha_vacunas;
}

public void setFecha_vacunas(Date fecha_vacunas) {
	this.fecha_vacunas = fecha_vacunas;
}

public PieChartModel getEstadisticaGraf() {
	return estadisticaGraf;
}

public void setEstadisticaGraf(PieChartModel estadisticaGraf) {
	this.estadisticaGraf = estadisticaGraf;
}

public CartesianChartModel getRepBarra() {
	return repBarra;
}
public void setRepBarra(CartesianChartModel repBarra) {
	this.repBarra = repBarra;
}
// Método que permite obtener las vacunas que se tiene en el carné
// retorna el listado de vacunas que se tiene en el carné
public List<SelectItem> getListaVacunasCarnetSI() {
	List<SelectItem> listadoSI = new ArrayList<SelectItem>();
	List<TblVacunasCarnet> listvacunacarnet = mVacunacion
			.findAllVacunasCarnet();
	try {
		for (TblVacunasCarnet c : listvacunacarnet) {

		SelectItem si = new SelectItem(c.getIdVc(), c

.getTblVacunaDosi().getTblUnionDistribucionVacunaN()
				.getTblStockVacuna().getNombreVacuna()
							+ "------->>"+
			c.getTblVacunaDosi().getTblNumDosi().getNombre());
			listadoSI.add(si);
		}
```

```
		} catch (Exception e) {
			FacesContext.getCurrentInstance().addMessage(null,
					new FacesMessage(e.getMessage()));
			e.printStackTrace();
		}
		return listadoSI;
	}

// Método para mostrar la estadística en grafica de barras
//este método permite mostrar todos los niños que han sido vacunados en los años mencionados, además posteriormente se verá como ingresamos el año podemos ver esta información, en este caso solo mostramos de los años 2012, 2013, 2014
	public CartesianChartModel getBarraVacunasAnuales() {
		// Variable para mostrar la estadística en gráfica de barra
		repBarra = new CartesianChartModel();
		List<TblCarnet> car = mVacunacion.findAllCarnet();
		List<TblVacunadosisConsulta> vacunas = mVacunacion
				.findAllVacunaDosisConsulta();
		// Variables poder acumular la cantidad de niños y niñas que han sido vacunados en el centro de salud
		int con2012F = 0;
		int con2012M = 0;
		int con2013M = 0;
		int con2013F = 0;
		int con2014M = 0;
		int con2014F = 0;
		// Variables que sirven como bandera para ver si ya ha sido vacunado o no
		boolean band2012 = false;
		boolean band2013 = false;
		boolean band2014 = false;
		//con esto capturamos la fecha actual y podemos transformar a un nuevo formato
		Calendar fin = Calendar.getInstance();
		//recorremos la lista de carné obtenida anteriormente
		for (TblCarnet tblCar : car) {
			band2012 = false;
			band2013 = false;
			band2014 = false;
		//recorremos la lista de vacunas  que tiene cada carné para verificar si hay vacunas que han sido administradas en el año solicitado
			for (TblVacunadosisConsulta tblCon : vacunas) {
				if (tblCar.getIdCarnet().equals(
						tblCon.getTblCarnet().getIdCarnet())
						&& tblCon.getEstado().equals("S")) {
					fin.setTime(tblCon.getFechaControl());

					if (fin.get(Calendar.YEAR) == 2012 &&
					band2012==false) {
						band2012=true;
				if tblCar.getTblPaciente().getGenero().equals("F")) {
							con2012F++;
						} else {
							con2012M++;
						}
```

```java
                    }
                if (fin.get(Calendar.YEAR) == 2013 && band2013==false) {
                        band2013=true;
                if (tblCar.getTblPaciente().getGenero().equals("F")) {
                            con2013F++;
                        } else {
                            con2013M++;
                        }

        }
        if (fin.get(Calendar.YEAR) == 2014 && band2014==false) {
            band2014=true;
            if (tblCar.getTblPaciente().getGenero().equals("F")) {
                            con2014F++;
                        } else {
                            con2014M++;
                        }

                    }

                }
            }
        }
    // Variables para crear la barra dependiendo del género en este caso
        ChartSeries boys = new ChartSeries();
        boys.setLabel("Niños");

        boys.set("2012", con2012M);
        boys.set("2013", con2013M);
        boys.set("2014", con2014M);

        ChartSeries girls = new ChartSeries();
        girls.setLabel("Niñas");

        girls.set("2012", con2012F);
        girls.set("2013", con2013F);
        girls.set("2014", con2014F);
    // Agregamos al modelo de barrar cada una de la barras con los datos tomados
    a mostrar.
        repBarra.addSeries(boys);
        repBarra.addSeries(girls);
        return repBarra;
    }

// Método que busca los pacientes que no han sido aplicadas las vacunas con
// la dosis seleccionada
public void AccionCargarDatos() {
    List<TblPaciente> listap = new ArrayList<TblPaciente>();
    try {
        TblVacunasCarnet vca = mVacunacion.findVacunasCarnetWhere(cod_vc);
            List<TblVacunadosisConsulta> vacc = vca
                    .getTblVacunadosisConsultas();
            if (vacc.size() > 0) {
                for (TblVacunadosisConsulta vacu_consu : vacc) {
```

```
                                        TblCarnet carne = vacu_consu.getTblCarnet();
                                        TblPaciente paciente = carne.getTblPaciente();
                                        listap.add(paciente);
                                }
                                lista_pac = listap;
                } else {
                FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage(FacesMessage.SEVERITY_INFO, "Informe:","No    hay pacientes
que tenga aplicada la vacuna y dósis especficada..!!"));
                        }
} catch (Exception e) {
        FacesContext.getCurrentInstance().addMessage(null,new
FacesMessage(FacesMessage.SEVERITY_ERROR,"Error:","Problemas al obtener pacientes
con estos detalles"));
                        e.printStackTrace();
        }
}
// Método que permite mostrar el detalle de la edad en el campo de texto una
// vez seleccionado la vacuna con su dosis, esto se utilizara en el reporte
//que se obtiene la lista de pacientes que han sido administrados una determinada
Dosis de una vacuna en un rango de edad seleccionado
public void AccionMostrarDetalle() {
        TblVacunasCarnet vca;
        try {
                vca = mVacunacion.findVacunasCarnetWhere(cod_vc);
                        detalle_edad = vca.getTblVacunaDosi()
                        .getTblUnionDistribucionVacunaN().getTblDistribucionEdade()
                                        .getDetalleDistribucion();
                } catch (Exception e) {
                        FacesContext.getCurrentInstance().addMessage(null,new
                        FacesMessage(FacesMessage.SEVERITY_ERROR,"Error:","Problemas al
                        obtener el detalle de la edad con vacuna y dósis
                        seleccionada"));
                        e.printStackTrace();
                }
        }
// Método para graficar el parte estadístico del porcentaje de niños y niñas que
existe en el centro de salud
        public PieChartModel getPieModel() {
                // List<TblPaciente> paciente = mVacunacion.findAllPaciente(miLogin
                // .getU_cod_esta().toString());
                List<TblPaciente> paciente = mVacunacion.findAllPaciente("1");
                int conM = 0;
                int conF = 0;
                for (TblPaciente tblp : paciente) {
                        if (tblp.getGenero().equals("F")) {
                                conF++;
                        } else {
                                conM++;
                        }
                }
                estadisticaGraf = new PieChartModel();
                estadisticaGraf.set("Niñas", conF);
                estadisticaGraf.set("Niños", conM);
                return estadisticaGraf;
```

```
        }

// Método que permite mostrar detalle cuando seleccionamos una determinada
// barra del Gráfico estadístico
//este metodo se produce en el evento de selección del ítem (barra)
public void itemSelect(ItemSelectEvent event) {
        FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Item Seleccionado :: "," Detalle:" + event.getSeriesIndex());
        FacesContext.getCurrentInstance().addMessage(null, msg);
}

// Método para capturar el objeto cuando se expande el p:rowToggler del datatable de
// los pacientes que tienen vacunas faltantes hasta la fecha actual
//esto nos muestra un p:rowExpansion con los detalles de la vacunas
//que están retrasados y las fechas que les tocaba dicha vacuna
//todo esto nos permite hacer los diferentes componentes de primefaces
public void rowEditListener(ToggleEvent event) {
        Visibility visibility = event.getVisibility();
        TblPaciente depar = (TblPaciente) event.getData();
        long ed = 0;
        try {
                ed = ope.calcularEdadEnMeses(depar.getFechaNacimientoPaciente());
                        int carnet = mVacunacion.findCarnetByIdHistoria(depar
                                        .getHistoriaClinica());
                        listavc = mVacunacion.findListaConsultasVacunas(carnet, new
                        Date());
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        ope.MSG(depar.getHistoriaClinica() + "      Es visible>>>>" + visibility
                                + "    con edad>>>>" + ed + "  TAMANIO DE LA ISTA>>>>>"
                                + listavc.size());
}

//Método que sirve para listar los pacientes que tenemos en nuestro centro de
Salud
// para ello se captura la sesión del ingreso de usuario ya que al iniciar la
sesión
//el usuario selecciona a que centro de salud pertenece y me muestra únicamente
los pacientes de dicho centro médico
//esto se utiliza para el reporte de mostrar los pacientes que faltan vacunas por
aplicar.
public List<TblPaciente> getListaPacientes() {
                return
                mVacunacion.findAllPaciente(miLogin.getU_cod_esta().toString());
}

//método que permite capturar los valores seleccionados en el p:pickList en el
//evento de selección, esto nos ayuda para ver los años de los cuales se va a
mostrar
//el reporte estadístico
public void onTransfer(TransferEvent event) {
                lineChar = getBarraVacunasAnuales();
                StringBuilder builder = new StringBuilder();
```

```
        for (Object item : event.getItems()) {
              builder.append(item.toString());
        }

        FacesMessage msg = new FacesMessage();
        msg.setSeverity(FacesMessage.SEVERITY_INFO);
        msg.setSummary("Año Transferido");
        msg.setDetail(builder.toString());

        FacesContext.getCurrentInstance().addMessage(null, msg);

  }
```